Práctica 2

Paradigmas de programación

El objetivo de esta práctica es explicar de manera detallada cómo funciona el programa.

ESTRUCTURA DEL CÓDIGO

env/ (Entorno Virtual): Este directorio utiliza un entorno virtual de Python. Al activar este entorno antes de correr el programa, se garantiza la consistencia y reproducibilidad de la ejecución.

src/ (Código Fuente): Este directorio comúnmente alberga el código fuente principal de la aplicación.

pycache: Guarda el bytecode compilado de los módulos .py en este directorio para acelerar la importación en ejecuciones posteriores.

static/: Contiene archivos estáticos utilizados por una interfaz web, como hojas de estilo CSS, archivos JavaScript, imágenes u otros recursos que se sirven directamente al navegador sin procesamiento dinámico.

templates/: Si el programa tiene una interfaz web, este directorio contendría los archivos de plantillas que definen la estructura y presentación de las páginas web dinámicas generadas por la aplicación.

biblioteca_web.py: Es la implementación de una interfaz web para la biblioteca.

biblioteca.py: Contiene la lógica principal del sistema de gestión de la biblioteca.

memory_management.py: Contiene código relacionado con la gestión de la memoria utilizada por la aplicación.

Makefile.txt: Está diseñado para automatizar tareas dentro del proyecto.

requirements.txt: Este archivo es crucial para gestionar las dependencias del proyecto. Enumera todos los paquetes (bibliotecas) de Python que el proyecto necesita para funcionar correctamente.

biblioteca_web.py

```
from flask import Flask, request, jsonify, render_template
from memory_management import memory_management
from biblioteca import Book, DigitalBook, Member, Library, Genre

app = Flask(__name__)
library = Library()
# Cargar datos al iniciar la aplicación
library.load_library_from_file("library.json")
library.load_members_from_file("members.json")
```

- Importa las clases necesarias de Flask y los módulos propios
- Crea una instancia de Flask (app) y de la biblioteca (library)

- Carga los datos iniciales desde archivos JSON

```
@app.route('/')
def index():
    return render_template('index.html')
```

- Muestra la página principal (index.html) cuando se accede a la raíz del sitio

```
@app.route('/books', methods=['GET'])
def get_books():
    books = [book.to_dict() for book in library.books]
    return jsonify(books)
```

- Devuelve una lista JSON con todos los libros

```
@app.route('/books', methods=['POST'])
def add_book():
    data = request.json
    is_digital = data.get('is_digital', False)
    if is_digital:
        book = DigitalBook(...)
    else:
        book = Book(...)
    library.add_book(book)
    library.save_library_to_file("library.json")
    memory_management.display_memory_usage()
    return jsonify(book.to_dict()), 201
```

- Crea un libro (digital o físico) según los datos recibidos
- Lo agrega a la biblioteca y guarda los cambios
- Muestra el uso de memoria y devuelve el libro creado

```
@app.route('/members', methods=['GET'])
def get_members():
    members = [member.to_dict() for member in library.members]
    return jsonify(members)
```

- Devuelve una lista JSON con todos los miembros

```
@app.route('/members', methods=['POST'])
def add_member():
    data = request.json
    member = Member(data['id'], data['name'])
    library.add_member(member)
```

```
    library.save_members_to_file("members.json")
    memory_management.display_memory_usage()
    return jsonify(member.to_dict()), 201
```

- Crea un nuevo miembro y lo agrega a la biblioteca
- Guarda los cambios y muestra el uso de memoria

```
@app.route('/issue_book', methods=['POST'])
def issue_book():
    data = request.json
    library.issue_book(data['book_id'], data['member_id'])
    library.save_library_to_file("library.json")
    library.save_members_to_file("members.json")
    memory_management.display_memory_usage()
    return jsonify({"message": "Libro prestado satisfactoriamente!"})
```

- Registra el préstamo de un libro a un miembro
- Guarda los cambios en ambos archivos

```
@app.route('/return_book', methods=['POST'])
def return_book():
    data = request.json
    library.return_book(data['book_id'], data['member_id'])
    library.save_library_to_file("library.json")
    library.save_members_to_file("members.json")
    memory_management.display_memory_usage()
    return jsonify({"message": "Libro devuelto satisfactoriamente!"})
```

- Registra la devolución de un libro
- Guarda los cambios en ambos archivos

```
@app.route('/save', methods=['POST'])
def save():
    library.save_library_to_file("library.json")
    library.save_members_to_file("members.json")
    memory_management.display_memory_usage()
    return jsonify({"message": "Datos guardados exitosamente!"})
```

- Guarda los datos actuales en los archivos JSON

```
@app.route('/load', methods=['POST'])
def load():
    library.load_library_from_file("library.json")
    library.load_members_from_file("members.json")
```

```
    memory_management.display_memory_usage()
    return jsonify({"message": "Datos cargados exitosamente!"})
```

- Recarga los datos desde los archivos JSON

```
@app.route('/genres', methods=['GET'])
def get_genres():
    genres = Genre.all_genres()
    return jsonify(genres)
```

- Devuelve una lista de todos los géneros disponibles

```
if __name__ == '__main__':
    app.run(debug=True)
```

- Inicia el servidor Flask en modo debug cuando se ejecuta directamente

Este código implementa una API de biblioteca usando Flask que permite gestionar libros (tanto físicos como digitales) y miembros mediante operaciones RESTful. La aplicación carga automáticamente los datos desde archivos JSON al iniciar y guarda los cambios después de cada modificación. La arquitectura modular permite distinguir entre libros tradicionales y digitales (estos últimos con atributos específicos como formato de archivo), proporcionando una solución completa y escalable para la administración de bibliotecas con persistencia de datos.

biblioteca.py

Este módulo define las clases principales para gestionar una biblioteca, incluyendo libros (físicos y digitales), miembros y operaciones básicas. A continuación, se explica su estructura clave:

Clase Genre (Géneros literarios):

```
class Genre:
    FICTION = "Ficcion"
    NON_FICTION = "No Ficcion"
    SCIENCE = "Ciencia"
    HISTORY = "Historia"
    FANTASY = "Fantasia"
    BIOGRAPHY = "Biografia"
    OTHER = "Otro"

    @classmethod
    def all_genres(cls):
        return [cls.FICTION, cls.NON_FICTION, cls.SCIENCE,
                cls.HISTORY, cls.FANTASY, cls.BIOGRAPHY, cls.OTHER]
```

- Proporciona una enumeración de géneros literarios.

Clase Book (Libro Base):

```
class Book:
    def __init__(self, book_id, title, author, publication_year, genre, quantity):
        self.id = book_id
        self.title = title
        self.author = author
        self.publication_year = publication_year
        self.genre = genre
        self.quantity = quantity
        memory_management.increment_heap_allocations(1)

    def __del__(self):
        memory_management.increment_heap_deallocations(1)

    def to_dict(self):
        return {
            "id": self.id,
            "title": self.title,
            "author": self.author,
            "publication_year": self.publication_year,
            "genre": self.genre,
            "quantity": self.quantity
        }

    @staticmethod
    def from_dict(data):
        return Book(data["id"], data["title"], ...)
```

- Constructor con parámetros básicos de un libro
- **to_dict()** para serialización
- **from_dict()** para deserialización
- Manejo de memoria en creación/eliminación

Clase DigitalBook(Herencia de Book):

```
class DigitalBook(Book):
    def __init__(self, book_id, title, ..., file_format):
        super().__init__(book_id, title, ...)
        self.file_format = file_format

    def to_dict(self):
        data = super().to_dict()
        data["file_format"] = self.file_format
        return data

    @staticmethod
```

```
    def from_dict(data):
        return DigitalBook(data["id"], data["title"], ..., data["file_format"])
```

- Extiende **Book** añadiendo:
  - Atributo file_format (PDF, EPUB, etc.)
  - Sobreescribe métodos de serialización para incluir el nuevo campo

Clase Member(Miembros):

```
class Member:
    def __init__(self, member_id, name):
        self.id = member_id
        self.name = name
        self.issued_books = []
        memory_management.increment_heap_allocations(1)

    def to_dict(self):
        return {
            "id": self.id,
            "name": self.name,
            "issued_books": self.issued_books
        }
```

- Información básica del miembro
- Lista de IDs de libros prestados (**issued_books**)

Class Library (Núcleo del sistema):

```
class Library:
    def __init__(self):
        self.books = []
        self.members = []

    def add_book(self, book):
        self.books.append(book)
        print("Libro agregado exitosamente!")

    def issue_book(self, book_id, member_id):
        book = self.find_book_by_id(book_id)
        member = self.find_member_by_id(member_id)
        if book and member and book.quantity > 0:
            book.quantity -= 1
            member.issued_books.append(book_id)
```

- Añadir libros/miembros
- Préstamos (issue_book) y devoluciones (return_book)
- Búsquedas por ID

- Persistencia en JSON

Persistencia en JSON:

```
def save_library_to_file(self, filename):
    with open(filename, 'w', encoding='UTF-8') as file:
        json.dump([book.to_dict() for book in self.books], file)

def load_library_from_file(self, filename):
    with open(filename, 'r', encoding='UTF-8') as file:
        data = json.load(file)
        self.books = [DigitalBook.from_dict(item) if "file_format" in item
                      else Book.from_dict(item) for item in data]
```

- **Guardado:** Convierte objetos a diccionarios y los escribe en JSON
- **Carga:** Reconstruye objetos desde JSON, diferenciando entre libros normales y digitales

Función Principal (main):

```
def main():
    library = Library()
    library.load_library_from_file("library.json")

    while True:
        print("\n1. Agregar libro\n2. Mostrar libros\n...")
        choice = input("Opción: ")

        if choice == "1":
            # Lógica para agregar libro
        elif choice == "2":
            library.display_books()
        # ... otras opciones ...
```

1. Carga datos iniciales
2. Muestra menú interactivo
3. Procesa opciones del usuario
4. Guarda al salir

Implementa un gestor de biblioteca con clases para libros (físicos/digitales) y miembros. Permite:

- Agregar/buscar libros y usuarios
- Gestionar préstamos y devoluciones
- Guardar datos en JSON automáticamente
- Interfaz de menú para operaciones

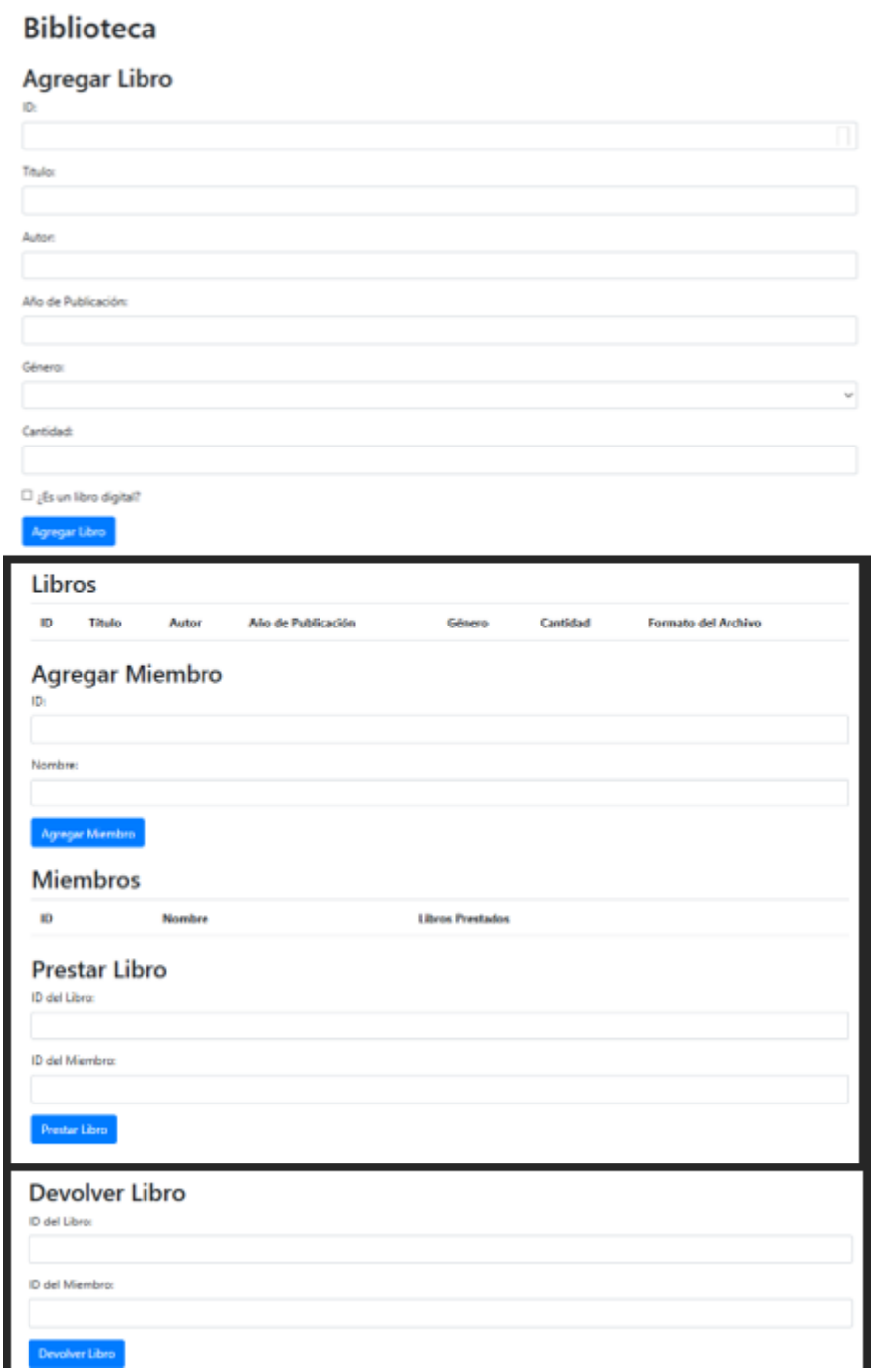
Usa POO con herencia (libros digitales), serialización JSON y gestión de memoria.

Funcionamiento del programa

El funcionamiento del programa sigue este flujo:

1. **Inicio**:
    - Al ejecutar, carga los datos existentes de **library.json** y **members.json**
    - Muestra un menú interactivo con 8 opciones
2. **Operaciones principales**:
    - *Libros*:
        1. Agregar nuevos (físicos/digitales)
        2. Mostrar todos disponibles
        3. Buscar por ID
    - *Miembros*:
        1. Registrar nuevos
        2. Consultar existentes
        3. Ver sus libros prestados
3. **Préstamos**:
    - Un miembro puede tomar prestado un libro (reduce stock)
    - Al devolverlo, se incrementa el stock
4. **Persistencia**:
    - Cada cambio importante guarda automáticamente en JSON
    - Datos persisten entre ejecuciones del programa
5. **Ejemplo de flujo**:
    1. Usuario selecciona "Agregar libro"
    2. Ingresa datos (título, autor, etc.)
    3. El sistema valida y guarda el libro
    4. Automáticamente actualiza **library.json**
    5. Puede prestarlo seleccionando "Préstamo"
6. **Salida**:
    - Al elegir "Guardar y salir", asegura la última versión en disco
    - Todos los cambios quedan persistentes para la próxima ejecución

Visualización



El sistema mantiene consistencia entre:

- Objetos en memoria
- Archivos JSON (backup)
- Interfaz de usuario (menú)

Permitiendo gestión completa sin pérdida de datos.

Repositorio: https://github.com/nidoco/PortafolioParadigmas/tree/master/content/practica2